Acelerando Algoritmos de Machine Learning em Ambientes não Estacionários Usando OpenMP

Camilla Magela[1], **Lucas Pinheiro**[1], **Thiago Damasceno**[1]
Marcelo Zamith[1], **Raul S. Ferreira**[1,2]

[1]DCC – Universidade Federal Rural do Rio de Janeiro (UFRRJ)
Nova Iguaçu – RJ – Brasil

[2]PESC/COPPE - Programa de Engenharia de Sistemas e Computação
Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro – RJ – Brazil

{camilla.magela, mzamith, raulsf}@ufrrj.br

Abstract. *Non-stationary environments suffers from a problem denominated concept-drift, that is, data distribution changes through the time. Solve such problems is extremely important in artificial intelligence field. Generally, algorithms that deal with concept-drift are applied in stream scenarios such as, robotics, image classification and sensors. However, to detect these changes, are necessary statistical procedures capable of support learning algorithms to adapt to distribution drifts. Since this scenarios have a vast amount of incoming data, the processing speed is crucial. Taking into account the mentioned challenges, we propose a parallelized image classification approach in non-stationary environments using the OpenMP tool. We show that with a few threads is possible to improve the entire learning process up to 355%. We show how we developed the entire parallelism, the efficiency and the speed-up achieved over the sequential version using a real dataset about images.*

Resumo. *Ambientes não estacionários sofrem de um problema conhecido como concept-drift, isto é, a distribuição dos dados muda de acordo com o tempo. Resolver tal problema é muito importante na área da inteligência artificial. Geralmente, algoritmos que lidam com o problema de concept-drift são aplicados em cenários de stream de dados, por exemplo, em robótica, classificação de imagens, sensores entre outros. Entretanto, para detectar essas mudanças, são necessários procedimentos estatísticos capazes de ajudar algoritmos de aprendizagem a se adaptarem as mudanças da distribuição. Por conta do cenário de stream de dados possuir uma grande quantidade de entrada de dados, a velocidade de processamento é crucial. Levando em conta os desafios mencionados, nós propomos uma abordagem paralelizada de classificação de imagens em ambientes não estacionários usando a ferramenta OpenMP. Mostramos que com poucas threads é possível melhorar a velocidade de todo o processo de aprendizado em até 355%. Mostramos como desenvolvemos o paralelismo, a eficiência e o speed-up alcançados sobre a versão sequencial em uma base de dados real.*

1. Introdução

Em ambientes não estacionários é comum que ocorra o problema conhecido como *concept-drift*. *Concept-drift* é a mudança da distribuição dos dados ao longo do tempo [Widmer e Kubat 1996]. Dessa forma, algoritmos de aprendizado de máquina devem aprender imediatamente com as alterações que ocorrem na distribuição dos dados, buscando uma melhor representação das classes dentre os dados recém chegados.Além disso, os modelos para lidar com este tipo de domínio de *stream* de dados devem não só ter uma boa acurácia e boa adaptabilidade à pequenas mudanças mas também devem ser rápidos o suficiente para dar a resposta necessária dentro de um ambiente dinâmico e com vasta quantidade de dados [Olorunnimbe et al. 2017].

Existem várias tarefas que podem ser realizadas por algoritmos de aprendizado de máquina, uma delas é a classificação. Neste contexto, existem soluções de classificação de imagens online, onde o modelo de aprendizado se adapta conforme a distribuição das imagens vai se alterando gradualmente [Qi et al. 2009]. Além disso, [Camps-Valls e Bruzzone 2005] mostraram que técnicas baseadas em *kernel*, como o *Support Vector Machine* (SVM) [Boser et al. 1992], funcionam muito bem para problemas de classificação de imagens.

O SVM é um algoritmo de aprendizado supervisionado usado para classificação e análise de regressão. O que o SVM faz é encontrar uma linha de separação, através do ajuste de duas variáveis, $Gamma$ e C. Assim, o algoritmo cria um hiperplano entre dados de duas classes. Essa linha busca maximizar a distância entre os pontos mais próximos em relação a cada uma das classes. A complexidade do SVM com kernel linear, usado neste trabalho, é $O(nk)$ onde n é o número de *features* e k o número de amostras .

Métodos supervisionados funcionam bem, quando apoiados por métodos estatísticos a fim de lidar com o problema de *concept-drift* [Capo et al. 2014]. Esta abordagem retém as melhores instâncias dentro de uma distribuição de dados ao longo do *stream* enquanto descarta as instâncias que podem atrapalhar o classificador.

Desta forma, propusemos uma abordagem paralela onde um classificador supervisionado, no caso o SVM, classifica as novas imagens do *stream* enquanto o método estatístico *Kernel Density Estimation* (KDE) [Parzen 1962] na versão multivariante, seleciona as imagens mais representativas já classificadas pelo SVM, para o próximo *batch* de imagens que chegará no tempo $t+1$. Assim, reduz-se a quantidade de dados utilizados na etapa de treinamento em $t+1$ enquanto detecta-se as instâncias mais representativas na distribuição dos dados.

KDE é uma técnica de estimativa de densidade usada para normalizar e suavizar a distribuição de um determinado conjunto de dados. É bastante usado em várias aplicações, desde ajudar a selecionar estimadores [Lacour et al. 2017] até aplicações *web* [Ferreira e Mello]. O KDE encontra as regiões de maior densidade da distribuição, contendo as instâncias mais relevantes. A complexidade do KDE multivariante com kernel Gaussiano, usado neste trabalho, é $O(n^2k)$ onde n é o número de instâncias e k o número de dimensões.

A demanda por processamento dos algoritmos usados neste trabalho eleva a necessidade de paralelismo, para que o tempo de processamento de todo o fluxo seja reduzido como consequência. Dentre as ferramentas disponíveis para paralelizar recursos compu-

tacionais, existe o OpenMP [Wang et al. 2010], que é uma API de programação paralela disponível em C/C++. A interface OpenMP foi escolhida para dar suporte ao paralelismo em memória compartilhada por conta de seu nível de abstração.

2. Experimentos

Cinco partes compõem o fluxo de classificação de imagens em ambientes dinâmicos. Processamento da base de dados, escolha dos melhores valores dos parâmetros do SVM, treino, classificação das imagens e processamento do KDE. Dessas cinco partes, o treino e a classificação não foram paralelizadas, pois nestes casos, o tempo de criação e processamento das *threads* acabaria por piorar o desempenho da aplicação.

O primeiro passo no fluxo é o tratamento da base de dados que é salvo em uma matriz coluna, onde cada matriz corresponde a uma imagem. Em seguida, a matriz é dividida em partes ou *batches*. A quantidade de *batches* é correspondente à quantidade de testes que serão realizados no total. O primeiro *batch* é utilizado para escolha dos parâmetros do SVM: C e $Gamma$. Para isso, é realizado uma validação cruzada combinando 8 valores para C e 8 valores para $Gamma$. O melhor valor de cada variável será usado de forma fixa no fluxo de dados.

Para os testes, a base de dados foi dividida em 20 *batches* que simulam a entrada de novos dados. O modelo treinado com os dados iniciais é utilizado para a classificar o *batch* de imagens não rotuladas. Em seguida, o *batch* recém classificado é passado para o algoritmo *KDE*. O KDE calcula as imagens que melhor representam a classe para qual foram classificadas. Uma certa porcentagem de imagens com maior densidade é retornada desse processo. As imagens retornadas serão usadas para treinar o SVM no *batch* seguinte, criando um novo modelo e sendo utilizado pra classificar o próximo *batch* de imagens não rotuladas. O processo se repete até o último *batch*.

A base de dados é composta de 9908 imagens, sendo estas divididas em duas classes: Blusas femininas e óculos escuros. A classe ”blusas”é composta de 4976 imagens e a classe ”óculos”composta de 4932 imagens. Cada uma das imagens possui um tamanho de 30x30 pixeis e para cada pixel contido na imagem é associado o valor das variáveis R, G e B, que correspondem aos níveis de vermelho, verde e azul da imagem respectivamente, totalizando 2700 atributos para cada imagem. Neste trabalho abordamos o desempenho de todo o fluxo de classificação na versão sequencial e paralelizado.

3. Resultados

Os testes foram realizados em uma máquina com as seguintes configurações:

- Processador i5-4590 com 4 núcleos de 3.3 GHz
- 6M de cache
- 8Gb de memória

No processamento sequencial foram obtidos os seguintes tempos, expressos em segundos:

1. Processamento da base de dados: 8,7 s
2. Processamento do Grid Search: 2.688 s
3. Processamento do KDE: 148.370 s

As tabelas 1, 2 e 3 mostram os resultados alcançados com a estratégia de paralelismo, variando em 2, 4 e 8 *threads*. Para medir o grau de aproveitamento dos recursos computacionais foi calculada a eficiência, que consiste em uma divisão do *speed up* pelo número de cores da máquina.

Tabela 1. Paralelo com 2 *threads*

	Tempo(sec)	Speedup	Eficiência
Processamento Dataset	4,42	1,965	0,982
Selecao de Parametros	1.492,87	1,800	0,900
KDE	78,91	1,880	0,940

Tabela 2. Paralelo com 4 *threads*

	Tempo(sec)	Speedup	Eficiência
Processamento Dataset	2,41	3,605	0,901
Selecao de Parametros	887,54	3,029	0,757
KDE	44,71	3,318	0,829

Tabela 3. Paralelo com 8 *threads*

	Tempo(sec)	Speedup	Eficiência
Processamento Dataset	3,11	2,792	0,349
Selecao de Parametros	749,79	3,585	0,448
KDE	48,67	3,048	0,381

4. Conclusão

Observa-se que os *speedups* são quase lineares para 2 e 4 *threads* em todas as etapas. Esse desempenho não foi mantido com 8 threads, onde a eficiência das etapas ficou entre 34,90% e 44,80%. Além disso, olhando para o tempo de conclusão das etapas, os melhores resultados na etapa de seleção foram alcançados com 8 *threads*, com ganhos de até 355% no tempo de processamento. Por outro lado, quando observa-se a eficiência em conjunto com o tempo, os resultados com 4 *threads* foram melhores.

Esse resultado mostra que a estratégia de paralelismo é promissora desde que a quantidade de *threads* seja limitada a quantidade de cores físicos da máquina. Portanto, se temos mais processos que processadores, em um único nó, só teremos paralelismo se houver I/O, ou seja, enquanto processos estão aguardando I/O, outros podem disputar o processador. Caso contrário, os testes sugerem que as *threads* competem por recursos, que podem ser a cache e/ou os cores.

Referências

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM.

Camps-Valls, G. and Bruzzone, L. (2005). Kernel-based methods for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 43(6):1351–1362.

Capo, R., Sanchez, A., and Polikar, R. (2014). Core support extraction for learning from initially labeled nonstationary environments using compose. In *Neural Networks (IJCNN), 2014 International Joint Conference on*, pages 602–608. IEEE.

Ferreira, R. S. and Mello, C. E. W-sage: Ferramenta web para análise de dados geoespaciais.

Lacour, C., Massart, P., and Rivoirard, V. (2017). Estimator selection: a new method with applications to kernel density estimation. *Sankhya A*, 79(2):298–335.

Olorunnimbe, M. K., Viktor, H. L., and Paquet, E. (2017). Dynamic adaptation of online ensembles for drifting data streams. *Journal of Intelligent Information Systems*, pages 1–23.

Parzen, E. (1962). On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076.

Qi, G.-J., Hua, X.-S., Rui, Y., Tang, J., and Zhang, H.-J. (2009). Two-dimensional multi-label active learning with an efficient online adaptation model for image classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(10):1880–1897.

Wang, J., Hu, C., Zhang, J., and Li, J. (2010). Openmp compiler for distributed memory architectures. *Science China information sciences*, 53(5):932–944.

Widmer, G. and Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23(1):69–101.